(54) Postback input handling by server-side control objects

(57) One or more server-side control objects provide server-side processing of postback input received from client-side user interface elements in a request from a client. Such input may include without limitation postback data and postback events generated from a web page on a client and received by a server in an HTTP request. A hierarchy of server-side control objects can cooperate to process the postback input and to generate the resulting authoring language code, such as HTML. Server-side events can also result from the server-side processing of postback input. In addition to processing postback input, server-side processing may include, without limitation, state management of server-side control objects and binding data to and from server-side database. After the processing and generating operations, the server-side control object hierarchy is terminated until a subsequent request is received from the client.
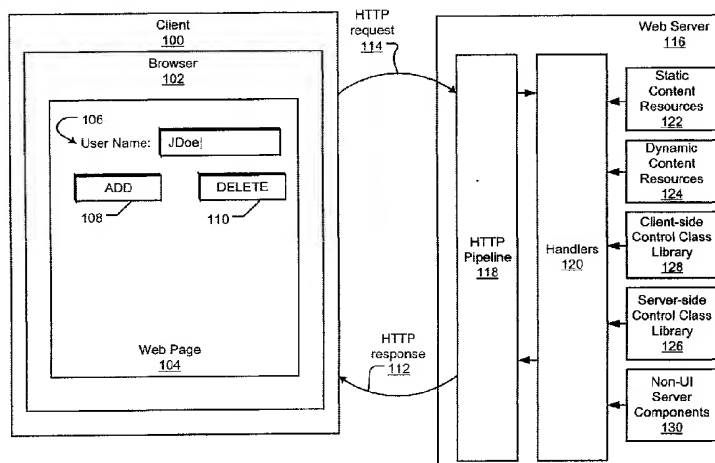
FIG. 1

EP 1 156 427 A2

Description

Technical Field

[0001]   The invention relates generally to a web server framework, and more particularly to server-side control objects that handle postback input received from a client.

Background of the Invention

[0002]   A typical web browser receives data from a web server that defines the appearance and rudimentary behavior of a web page for display on a client system. In a typical scenario, a user specifies a Uniform Resource Locator ("URL"), a global address of a resource on the World Wide Web, to access a desired web site. Generally, the term "resource" refers to data or routines that can be accessed by a program. An example URL is "http://www.microsoft.com/ms.htm". The first part of the example URL indicates a given protocol (i.e., "http") to be used in the communication. The second part specifies the domain name (i.e., "www.microsoft.com") where the resource is located. The third part specifies the resource (i.e., a file called "ms.htm") within the domain. Accordingly, a browser generates an HTTP (HyperText Transport Protocol) request associated with the example URL to retrieve the data associated with ms.htm file within the www.microsoft.com domain. A web server hosting the www.microsoft.com site receives the HTTP request and returns the requested web page or resource in an HTTP response to the client system for display in the browser.

[0003]   The "ms.htm" file of the example above includes static HTML (HyperText Markup Language) code. HTML is a plain-text authoring language used to create documents (e.g., web pages) on the World Wide Web. As such, an HTML file can be retrieved from a web server and displayed as a web page in a browser to present the rich graphical experience that users have come to expect while viewing information from the Internet. Using HTML, a developer can, for example, specify formatted text, lists, forms, tables, hypertext links, inline images and sounds, and background graphics for display in the browser. An HTML file, however, is a static file that does not inherently support dynamic generation of web page content.

[0004]   If dynamic content is to be displayed, such as a changing stock price or traffic information, a server-side application program is generally developed to handle the more complex client-server interaction. The server-side application program processes the HTTP request and generates the appropriate HTML code for transmission to the client in an HTTP response. An exemplary HTTP request may include parameters, such as data in query string or data from web-based forms. As such, a server-side application program can process these parameters and dynamically generate HTML code for transmission in an HTTP response to the client. An exemplary server-side application program may generate a document containing appropriate HTML code using a sequence of one or more formatted text write operations to a memory structure. Thereafter, the resulting document is transmitted to a client system in an HTTP response, where it is displayed as a web page in the browser.

[0005]   Developing a server-side application program can be a complex task requiring not only familiarity with normal HTML coding that is used to layout a web page, but also with programming basics, including one or more programming languages (e.g., C++, Perl, Visual Basic, or Jscript). Web page designers, on the other hand, are frequently graphics designers and editors, who may lack programming experience. Furthermore, simplifying complex web page development can speed the development of new web content by any developer. Generally, development of a custom server-side application also requires tremendous effort, so much, in fact, that developers are often disinclined to attempt it. It is desirable, therefore, to provide a development framework that allows a developer to dynamically create and process a web page with minimal programming.

[0006]   One approach to minimize the programming requirements of dynamic web page generation has been the Active Server Page (ASP) framework, provided by Microsoft Corporation. An ASP file typically includes Visual Basic or Jscript code, for example, to process an HTTP request that specifies the ASP file as the desired resource and, thereafter, to generate the resulting HTML code in a HTTP response to the client. Furthermore, an ASP file may reference pre-developed or third party client-side library components (e.g., client-side "ACTIVEX" controls) to ease a given application programming effort. However, in the current server-side application frameworks, the programming required to dynamically manage client-side user interface elements (e.g., text boxes, list boxes, buttons, hypertext links, images, sounds, etc.) within server-side applications can still require sophisticated programming skills and considerable effort. An unanswered problem exists in properly encapsulating programming required to process user interface elements, so as to allow the web page developer to focus on other aspects of the web page.

[0007]   More specifically, postback events and data from user interface elements are communicated from the client to the server for processing. Handling of such postback input is typically performed in custom server-side non-user-interface server components, which can require sophisticated user interface programming. It is desirable to encapsulate handling of postback input from a client-side user interface element in a server-side framework so as to simplify the programming requirements of associated non-user-interface server components.

Summary of the Invention

[0008] In accordance with the present invention, the above and other problems are solved by server-side processing of postback input received from client-side user interface elements in a request from a client. Such input may include without limitation postback data and postback events generated from a web page on a client and received by a server in an HTTP request. Furthermore, a hierarchy of member server-side control objects can cooperate to process the postback input and to generate the resulting authoring language code, such as HTML. After the processing and generating operations, the server-side control object hierarchy is terminated until a subsequent request from the client.

[0009] A method of performing server-side processing of postback input received from a client and associated with a client-side user interface element is provided. The postback input is examined to determine an identifier of a target server-side control object. The target server-side control object is identified based on the identifier of the target server-side control object. The postback input is passed to the target server-side control object and processed therein.

[0010] A computer program product providing server-side processing of postback input received from a client and associated with a client-side user interface element is provided. Server-side control objects are created in a server-side control object hierarchy. The server-side control object hierarchy includes a target server-side control object associated with the client-side user interface element. The postback data is passed to the target server-side control object. The postback input received by target server-side control object is processed therein. Authoring language data is generated from the plurality of server-side control objects to define a web page for display on a client.

[0011] An embodiment of a computer program product in accordance with the present invention includes a computer program storage medium readable by a computer system and encoding a computer program for executing a computer process providing server-side processing of postback input received from a client and associated with a client-side user interface element. An alternative embodiment of a computer program product in accordance with the present invention includes a computer data signal embodied in a carrier wave by a computing system and encoding a computer program for executing a computer process for providing server-side processing of postback input received from a client and associated with a client-side user interface element.

Brief Description of the Drawings

[0012] FIG. 1 illustrates a web server for dynamically generating web page content for display on a client in an embodiment of the present invention.

[0013] FIG. 2 illustrates a flow diagram of operations for processing and rendering client-side user interface elements using server-side control objects in an embodiment of the present invention.

[0014] FIG. 3 illustrates exemplary modules in a web server used in an embodiment of the present invention.

[0015] FIG. 4 illustrates an exemplary dynamic content resource (e.g., an ASP+ resource) in an embodiment of the present invention.

[0016] FIG. 5 illustrates an exemplary system useful for implementing an embodiment of the present invention.

[0017] FIG. 6 illustrates a process flow diagram representing processing of a page object in an embodiment of the present invention.

[0018] FIG. 7 illustrates a flow diagram of client-side operations relating to processing postback input in an embodiment of the present invention.

[0019] FIG. 8 illustrates a flow diagram of server-side operations for processing postback input in an embodiment of the present invention.

[0020] FIG. 9 illustrates a flow diagram of server-side operations for locating a control object in a control object hierarchy, responsive to a receipt of a control object path identifier, in an embodiment of the present invention.

Detailed Description of the Invention

[0021] An embodiment of the present invention includes server-side computer process creating one or more server-side control objects providing server-side processing of postback input. The server-side control objects logically correspond to client-side user interface elements and generate at a server the authoring language code to be used by a client-side browser to display and process a web page. The postback input may include, without limitation, postback data and postback events generated from a web page on a client and received by a server in an HTTP request. Furthermore, a hierarchy of server-side control objects can cooperate to process the postback input and to generate the resulting authoring language code, such as standard HTML or higher. Server-side events can also result from the server-side processing of postback input. In addition to processing postback input, server-side processing may include, without limitation, state management of server-side control objects and binding data to and from server-side database. After the processing and generating operations, the server-side control object hierarchy is terminated until a subsequent request is received from the client.

[0022] FIG. 1 illustrates a web server for dynamically generating web page content for display on a client in an embodiment of the present invention. A client 100 executes a browser 102 that displays a web page 104 on a display device of the client 100. The client 100 may include a client computer system having a display device, such as a video monitor. An "INTERNET EXPLORER" browser, marketed by Microsoft Corporation, is an example of a browser 102 in an embodiment of the present invention. Other exemplary browsers include without limitation "NETSCAPE NAVIGA-TOR" and "MOSAIC". The exemplary web page 104 includes a text box control 106 and two button controls 108 and 110. The browser 102 may receive HTML code in the HTTP response 112 from a web server 116 and displays the web page as described by the HTML code. Although HTML is described with reference to one embodiment, other authoring languages, including without limitation SGML (Standard Generalized Markup Language), XML (eXtensible Markup Language), and WML (Wireless Markup Language), which is an XML-based markup language, designed for specifying the content and user interfaces of narrowband wireless devices, such as pagers and cellular phones, are contemplated within the scope of the present invention. Furthermore, although standard HTML 3.2 is primarily disclosed herein, any version of HTML is supportable within the scope of the present invention.

[0023] The communications between the client 100 and the web server 116 may be conducted using a sequence of HTTP requests 114 and HTTP responses 112. Although HTTP is described with reference to one embodiment, other transport protocols, including without limitation S-HTTP, are contemplated within the scope of the present invention. On the web server 116, an HTTP pipeline module 118 receives an HTTP request 114, resolves the URL, and invokes an appropriate handler 120 for processing the request. In an embodiment of the present invention, a plurality of handlers 120 to handle different types of resources are provided on the web server 116.

[0024] For example, if the URL specifies a static content resource 122, such as an HTML file, a handler 120 accesses the static content resource 122 and passes the static content resource 122 back through the HTTP pipeline 118 for communication to the client 100 in an HTTP response 112. Alternatively, in an embodiment of the present invention, if the URL specifies a dynamic content resource 124, such as an ASP+ resource, a handler 120 accesses the dynamic content resource 124, processes the contents of the dynamic content resource 124, and generates the resulting HTML code for the web page 104. In an embodiment of the present invention, the resulting HTML code includes standard HTML 3.2 code. Generally, a dynamic content resource is a server-side declaration datastore (e.g., an ASP+ resource) that can be used to dynamically generate the authoring language code that describes a web page to be displayed on a client. The HTML code for the web page is then passed through the HTTP pipeline 118 for communication to the client 100 in an HTTP response 112.

[0025] During its processing, a handler 120 can also access libraries of pre-developed or third-party code to simplify the development effort. One such library is a server-side class control library 126, from which the handler 120 can instantiate server-side control objects for processing user interface elements and generating the resultant HTML data for display of a web page. In an embodiment of the present invention, one or more server-side control objects map to one or more user interface elements, visible or hidden, on the web page described in the dynamic content resource 124.

[0026] A second library, in contrast, is a client-side control class library 128, such as a library including "ACTIVEX" components from Microsoft Corporation. An "ACTIVEX" control is a COM ("Component Object Model") object that follows certain standards in how it interacts with its client and other components. A client-side "ACTIVEX" control, for example, is a COM-based component that can be automatically downloaded to a client and executed by a web browser on the client. Server-side ACTIVEX components (not shown) are COM-based components that may be implemented on a server to perform a variety of server-side functions, such as providing the server-side functionality of a stock price look-up application or database component. A more detailed discussion of ACTIVEX can be found in "Understanding ACTIVEX and OLE", David Chappell, Microsoft Press, 1996.

[0027] In contrast to "ACTIVEX" controls, a server-side control object in an embodiment of the present invention, being specified in a dynamic content resource 124, logically corresponds to a user interface element that is incorporated in the web page on the client. The server-side control object can also generate valid HTML code that can include, for example, an HTML tag and a locator referencing a given client-side "ACTIVEX" control. If the browser already has the code for the client-side "ACTIVEX" control within its storage system, the browser executes the "ACTIVEX" control within the web page on the client. Otherwise, the browser downloads the code for the "ACTIVEX" control from the resource specified by the locator and then executes the "ACTIVEX" control within the web page on the client. A server-side control object in an embodiment of the present invention can also raise events to a server-side "ACTIVEX" component used to implement a stock look-up application on the server.

[0028] A handler 120 also has access to one or more non-user-interface server components 130 that execute on the web server 116 or on another accessible web server. A non-user-interface server component 130, such as a stock price look-up application or database component, may be referenced in or associated with a dynamic content resource 124 that is processed by a handler 120. Server-side events raised by the control objects declared in the dynamic content resource 124 may be processed by server-side code, which calls appropriate methods in the non-user-interface server component 130. As a result, the processing provided by the server-side control objects simplifies the programming of the non-user-interface server component 130 by encapsulating the processing and generation of the user

interface elements of a web page, which allows the developer of the non-user-interface server component 130 to concentrate on the specific functionality of the application, rather than on user interface issues.

[0029] FIG. 2 illustrates a flow diagram of operations for processing and generating client-side user interface elements using server-side control objects in an embodiment of the present invention. In operation 200, the client transmits an HTTP request to the server. The HTTP request includes a URL that specifies a resource, such as an ASP+ resource. In operation 202, the server receives the HTTP request and invokes the appropriate handler for processing the specified resource. The ASP+ resource is read in operation 203. Operation 204 generates a server-side control object hierarchy based on the contents of the specified dynamic content resource (e.g., the ASP+ resource).

[0030] In operation 206, the member server-side control objects of the control object hierarchy perform one or more of the following operations: postback event handling, postback data handling, state management, and data binding. Postback events and data (collectively "postback input") from user interface elements are communicated from the client to the server for processing. A postback event, for example, may include without limitation a "mouse click" event from a client-side button element or a "data change" event from a client-side textbox element that is communicated to the server. Postback data, for example, may include without limitation text entered by a user in a text box element or an index of an item selected from a drop-down box.

[0031] In operation 208, each server-side control object in the hierarchy is called to generate (or render) data, such as HTML code, for display of client-side user interface elements in the web page. Note that, although the term "render" may be used to described the operation of displaying graphics on a user interface, the "render" is also used herein to describe the operation of generating authoring language data that can be interpreted by a client application, such as a browser, for display and client-side functionality. A more detailed discussion of the processing operation 206 and the rendering operation 208 is provided in association with FIG. 6. In one embodiment, calls to render() methods in individual control objects are performed using a tree traversal sequence. That is, a call to the render() method of a page object results in recursive traversal throughout appropriate server-side control objects in the hierarchy. Alternative methods for calling the render() methods for appropriate control objets may also be employed, including an event signaling or object registration approach. The parentheses designate the "render()" label as indicating a method, as compared to a data value.

[0032] In an embodiment of the present invention, the actual creation of the individual server-side control objects may be deferred until the server-side control object is accessed (such as when handling postback input, loading a state, rendering HTML code from the control object, etc.) in operations 206 or 208. If a server-side control object is never accessed for a given request, deferred control object creation optimizes server processing by eliminating an unnecessary object creation operation.

[0033] Operation 210 transmits the HTML code to the client in an HTTP response. In operation 214, the client receives the HTML code associated with a new web page to be displayed. In operation 216, the client system incorporates (e. g., displays) the user interface elements of the new page in accordance with the HTML code received from the HTTP response. It should be understood, however, that incorporation of a user-interface element may include non-display operations, such as providing audio or tactile output, reading and writing to memory, controlling the operation of scripts, etc. In operation 212, the server-side control object hierarchy is terminated. In one embodiment of the present invention, server-side control objects in the hierarchy are created in response to an HTTP request referencing an associated ASP+ resource, and destroyed subsequent to the rendering of authoring language data (e.g., HTML data). In an alternative embodiment, operation 212 may be performed after operation 208 and before operation 210.

[0034] FIG. 3 illustrates exemplary modules in a web server used in an embodiment of the present invention. The web server 300 receives an HTTP request 302 into the HTTP pipeline 304. The HTTP pipeline 304 may include various modules, such as modules for logging of web page statistics, user authentication, user authorization, and output caching of web pages. Each incoming HTTP request 302 received by the web server 300 is ultimately processed by a specific instance of an IHTTPHandler class (shown as handler 306). The handler 306 resolves the URL request and invokes an appropriate handler factory (e.g., a page factory module 308).

[0035] In FIG. 3, a page factory module 308 associated with the ASP+ resource 310 is invoked to handle the instantiation and configuration of the ASP+ resource 310. In one embodiment, an ASP+ resource 310 can be identified by designating a particular suffix (or file extension such as ".aspx") with a resource. When a request for a given ".aspx" resource is first received by the page factory module 308, the page factory module 308 searches the file system for the appropriate text file (e.g., the .aspx file 310). The file may contain text (e.g., authoring language data) or another data format (e.g., byte-code data or encoded data) that may later be interpreted or accessed by the server to service the request. If the physical file exists, the page factory module 308 opens the file and reads the file into memory. If the file cannot be found, the page factory module 308 returns an appropriate "file not found" error message.

[0036] After reading the ASP+ resource 310 into memory, the page factory module 308 processes the file content to build a data model of the page (e.g., lists of script blocks, directives, static text regions, hierarchical server-side control objects, server-side control properties, etc.). The data model is used to generate a source listing of a new object class, such as a COM+ ("Component Object Model") class, that extends the page base class. The page base class

includes code that defines the structure, properties, and functionality of a basic page object. In an embodiment of the present invention, the source listing is then dynamically compiled into an intermediate language and later Just-In-Time compiled into platform native instructions (e.g., X86, Alpha, etc.). An intermediate language may include general or custom-build language code, such as COM+ IL code, Java bytecodes, Modula 3 code, SmallTalk code, and Visual Basic code. In an alternative embodiment, the intermediate language operations may be omitted, so that the native instructions are generated directly from the source listing or the source file (e.g., the ASP+ resource 310). A control class library 312 may be accessed by the page factory module 308 to obtain predefined server-side control classes used in the generation of the control object hierarchy.

[0037]   The page factory module 308 outputs a page object 314, which is a server-side control object that corresponds to the web page 104 of FIG. 1. The page object 314 and its children (i.e., a text box object 318, a button object 320, and another button object 322) comprise an exemplary control object hierarchy 316. Other exemplary control objects are also contemplated in accordance with the present invention, including without limitation objects corresponding to the HTML controls in Table 1, as well as custom control objects. The page object 314 logically corresponds to the web page 104 of FIG. 1. The text box object 318 logically corresponds to the text box 106 in FIG. 1. Likewise, the button object 320 logically corresponds to the add button 108 in FIG. 1, and the button object 322 corresponds to the delete button 110 in FIG. 1. The page object 314 is hierarchically related to other control objects on the server. In one embodiment, a page object is a container object that hierarchically contains its children control objects. In an alternative embodiment, other forms of hierarchical relation may be employed, including a dependency relationship. In a more complex control object hierarchy with multiple levels of children, a child object can be a container object for other child objects.

[0038]   In the illustrated embodiment, the control objects in the control object hierarchy 316 are created and executed on the server 300, and each server-side control object logically corresponds to a user interface element on the client. The server-side control objects also cooperate to handle postback input from the HTTP request 302, to manage the states of server-side control objects, to perform data binding with server-side databases, and to generate authoring language data (e.g., HTML code) used to display a resulting web page at the client. The resulting authoring language data is generated (i.e., rendered) from the server-side control object hierarchy 316 and transmitted to the client in an HTTP response 324. For example, resulting HTML code may embody any valid HTML construct and may reference ACTIVEX-type control, JAVA applets, scripts, and any other web resources that yield client-side user interface elements (e.g., control buttons, text boxes, etc.) when processed by a browser.

[0039]   By virtue of declarations made in the ASP+ resource 310, server-side control objects may also access one or more non-user-interface server components 330 to provide interaction between the non-user-interface server component 330 and client-side user interface elements. For example, in response to postback input, server-side control objects can raise server-side events to server-side code registered for those events. The server-side code may be developed by the page developer and designated with the "runat=server" attribute. In turn, the server-side code calls methods in the non-user-interface server component 330. In this manner the non-user-interface server component 330 can interact with the user through client-side user interface elements without programming the code required to display and process these elements.

[0040]   FIG. 4 illustrates contents of an exemplary dynamic content resource in an embodiment of the present invention. In the illustrated embodiment, the file 400 contains plain-text declarations in an exemplary dynamic content resource format (e.g., ASP+). Each declaration provides instructions to a page compiler that reads the file 400, creates and invokes the appropriate server-side control objects, and ultimately combines the rendered HTML code for transmission to the client in an HTTP response.

[0041]   The first line of the file 400 includes a directive in the format:

    <%@ directive {attribute=value} %>

where *directive* may include without limitation "page", "cache", or "import". Directives are used by the page compiler when processing a dynamic content resource to determine such characteristics as buffering semantics, session state requirements, error handling schemes, scripting languages, transaction semantics, and import directions. Directives may be located anywhere within a page file.

[0042]   In the second line, <html> is a standard HTML starting tag, which is passed through to the resulting HTML code as a literal (i.e., without additional processing to render the resulting HTML code). In HTML, the <html> indicates the beginning of the HTML file and is paired with the closing tag on line 21, </html>, which is also a literal.

[0043]   A code declaration block is located at lines 3-10 of the file 400. Generally, server-side code declaration blocks define page object and control object member variables and methods that are executed on the server. In the format:

    <script runat = "server" [language = "*language*"][src = "*externalfile*"]>

        ....

    </script>

where the language and src parameters are optional. In an embodiment of the present invention, code declaration blocks are defined using <script> tags that contain a "runat" attribute having a value set to "server". Optionally, a

"language" attribute may be used to specify the syntax of the inner code. The default language may represent the language configuration of the overall page; however, the "language" attribute in the code declaration block allows a developer to use different languages within the same web page implementation including, for example, Jscript and PERL (Practical Extraction and Report Language). The <script> tag may also optionally specify a "src" file, which is an external file from which code is inserted into the dynamic content resource for processing by the page compiler. It should be understood that the disclosed syntax is used in one embodiment, however, alternative embodiments may employ different syntaxes within the scope of the present invention.

[0044] In FIG. 4, two subroutines are declared in Visual Basic format within the code declaration block: AddButton_Click and DeleteButton_Click. Both subroutines take two input parameters, "Source" and "E", and are executed on the server in response to an HTTP request when a client-side click event is detected on the corresponding button. In the AddButton_Click subroutine, the text in a UserName text box is concatenated on to the word "Add" and loaded into the Text data member of Message. In the DeleteButton_Click subroutine, the text in the UserName text box is concatenated on to the word "Delete" and loaded into the Text data member of Message. Although not shown in FIG. 4, member variables of server-side control objects may be declared in the code declaration block of the file 400. For example, using a Visual Basic syntax, the key word space "DIM" declares a data variable of a server-side control object.

[0045] A "code render block" (not shown) can also be included in a dynamic content resource. In an embodiment of the present invention, a code render block executes in a single "rendering" method that executes at page render time. A code render block satisfies the following format (although other formats are contemplated in alternative embodiments):

        <% InlineCode %>

where InlineCode includes self-contained code blocks or control flow blocks that execute on the server at page render time.

[0046] Inline expressions may also be used within a code render block, using the exemplary syntax:

        <%= InlineExpression %>

where the expression contained in an InlineExpression block is ultimately encompassed by a call to "Response.Write (InlineExpression)" in a page object, which writes the value resulting from InlineExpression into an appropriate place holder in the declaration. For example, InlineExpressions may be included in a code render block as follows:

        <font size = "<%=x%>" > Hi <%=Name%>, you are <%=Age%>!

which outputs a greeting and a statement about a person's age in a font stored in the value "x". The person's name and age are defined as strings in a code declaration block (not shown). The resulting HTML code is rendered at the server for transmission to the client in the HTTP response so as to include the values of the InlineExpressions at appropriate locations:

        <font size="12"> Hi Bob, your are 35!

[0047] On line 11 of file 400, <body> is a standard HTML tag for defining the beginning of the body of the HTML document. On line 20 of file 400, the closing tag, </body>, is also shown. Both the <body> and </body> are literals in an embodiment of the present invention.

[0048] Within the body section of the HTML file 400, on line 12, the starting tag, <form>, of an HTML form block is found in FIG. 4. The ending tag, </form>, of the form block is found on line 19 of the HTML file 400. An optional parameter "id" may also be included in the HTML control tag, <form>, to associate a given identifier with the form block, thereby allowing multiple form blocks to be included in a single HTML file.

[0049] On line 18 of file 400, a server-side label, identified by "Message", is declared. The "Message" label is used in the code declared at lines 5 and 8 of the file 400 to display a label on the web page.

[0050] Within a form block, three exemplary HTML control tags are shown, corresponding to the user interface elements 106, 108, and 110 of FIG. 1. The first user interface element is declared on line 13 of the file 400 corresponding to a textbox. The text literal "User Name:" declares a label positioned to the left of the textbox. The input tag with type="Text" declares a textbox server-side control object having an id equaling "UserName" as a server-side control object that renders a textbox client-side user interface element. Lines 15 and 16 of file 400 declare the client-side user interface elements shown as buttons 108 and 110 of FIG. 1, respectively. Note that the "OnServerClick" parameter specifies the appropriate sub-routine declared in the code declaration block of the file 400. As such, the server-side button control objects generated in response to the declarations in file 400 render the HTML code for the client-side buttons and an associated server-side script for implementing the button click events.

[0051] The textbox and buttons declared in file 400 are examples of HTML server control declarations. By default, all HTML tags within an ASP+ resource are treated as literal text content and are programmatically inaccessible to page developers. However, page authors can indicate that an HTML tag should be parsed and treated as an accessible server control declaration by marking it with a "runat" attribute with a value set to "server". Each server-side control object may optionally be associated with a unique "id" attribute to enable programmatic referencing of the corresponding control object. Property arguments and event bindings on server-side control objects can also be specified using de-

HTML Tag Name	Example	COM+ Class
<a>	 My Link 	AnchorButton
<img>	<img id = "MyImage" runat = server>	Image
	 	Label
<div>	<div id = "MyDiv" runat = server>Some contents</div>	Panel
<form>	<form id = "MyForm" runat = server> </form>	FormControl
<select>	<select id = "MyList" runat = server> <option>One</option> <option>Two</option> <option>Three</option> </select>	DropDownList
<input type = file>	<input id = "MyFile" type = file runat = server>	FileInput
<input type = text>	<input id = "MyTextBox" type = text>	TextBox
<input type = password>	<input id = "MyPassword" type = password>	TextBox
<input type = reset>	<input id = "MyReset" type = reset>	Button
<input type = radio>	<input id = "MyRadioButton" type = radio runat = server>	RadioButton
<input type = checkbox>	<input id = "MyCheck" type = checkbox runat = server>	CheckBox
<input type = hidden>	<input id ="MyHidden" type = hidden runat = server>	HiddenField
<input type = image>	<input type = image src = "foo.jpg" runat = server>	ImageButton
<input type = submit>	<input type = submit runat = server>	Button
<input type = button>	<input type = button runat = server>	Button
<button>	<button id = MyButton runat = server>	Button
<textarea>	<textarea id = "MyText" runat = server> This is some sample text </textarea>	TextArea

TABLE 1

[0053] In addition to standard HTML control tags, an embodiment of the present invention enables developers to create re-usable components that encapsulate common programmatic functionality outside of the standard HTML tag set. These custom server-side control objects are specified using declarative tags within a page file. Custom server-side control object declarations include a "runat" attribute with a value set to "server". Optionally, the unique "id" attribute may be specified to enable programmatic referencing of the custom control object. In addition, declarative name/value attribute pairs on a tag element specify property arguments and event bindings on a server-side control object. In-line template parameters may also be bound to a server-side control object by providing an appropriate "template" prefix

child-element to the parent server control object. A format for a custom server-side control object declaration is:

<servercntrlclassname id="OptionalName" [propertyname="propval"] runat=server/>

where servercntrlclassname is a name of an accessible server control class, OptionalName is a unique identifier of the server-side control object, and propval represents an optional property value in the control object.

[0054]   Using an alternative declaration syntax, XML tag prefixes may be used to provide a more concise notation for specifying server-side control objects within a page, using the following format:

<tagprefix:classname id = "OptionalName" runat = server/>

where tagprefix is associated with a given control name space library and classname represents a name of a control in the associated name space library. An optional propertyvalue is also supported.

[0055]   In summary, an embodiment of the present invention includes server-side control objects that are created and executed on the server to generate HTML code that is sent to a client. The HTML code may, for example, embody any valid HTML constructs and may reference ACTIVEX-type controls, JAVA applets, scripts, and any other web resources to produce user interface buttons and other user interface elements at the client. A user at the client may interact with these user interface elements, which logically correspond to the server-side control objects, and send a request back to the server. The server-side control objects are recreated on the server to process the data, events, and other characteristics of the user interface elements so as to generate the next round of HTML code to be transmitted in a response to the client.

[0056]   With reference to FIG. 5, an exemplary computing system for embodiments of the invention includes a general purpose computing device in the form of a conventional computer system 500, including a processor unit 502, a system memory 504, and a system bus 506 that couples various system components including the system memory 504 to the processor unit 500. The system bus 506 may be any of several types of bus structures including a memory bus or memory controller, a peripheral bus and a local bus using any of a variety of bus architectures. The system memory includes read only memory (ROM) 508 and random access memory (RAM) 510. A basic input/output system 512 (BIOS), which contains basic routines that help transfer information between elements within the computer system 500, is stored in ROM 508.

[0057]   The computer system 500 further includes a hard disk drive 512 for reading from and writing to a hard disk, a magnetic disk drive 514 for reading from or writing to a removable magnetic disk 516, and an optical disk drive 518 for reading from or writing to a removable optical disk 519 such as a CD ROM, DVD, or other optical media. The hard disk drive 512, magnetic disk drive 514, and optical disk drive 518 are connected to the system bus 506 by a hard disk drive interface 520, a magnetic disk drive interface 522, and an optical drive interface 524, respectively. The drives and their associated computer-readable media provide nonvolatile storage of computer readable instructions, data structures, programs, and other data for the computer system 500.

[0058]   Although the exemplary environment described herein employs a hard disk, a removable magnetic disk 516, and a removable optical disk 519, other types of computer-readable media capable of storing data can be used in the exemplary system. Examples of these other types of computer-readable mediums that can be used in the exemplary operating environment include magnetic cassettes, flash memory cards, digital video disks, Bernoulli cartridges, random access memories (RAMs), and read only memories (ROMs).

[0059]   A number of program modules may be stored on the hard disk, magnetic disk 516, optical disk 519, ROM 508 or RAM 510, including an operating system 526, one or more application programs 528, other program modules 530, and program data 532. A user may enter commands and information into the computer system 500 through input devices such as a keyboard 534 and mouse 536 or other pointing device. Examples of other input devices may include a microphone, joystick, game pad, satellite dish, and scanner. These and other input devices are often connected to the processing unit 502 through a serial port interface 540 that is coupled to the system bus 506. Nevertheless, these input devices also may be connected by other interfaces, such as a parallel port, game port, or a universal serial bus (USB). A monitor 542 or other type of display device is also connected to the system bus 506 via an interface, such as a video adapter 544. In addition to the monitor 542, computer systems typically include other peripheral output devices (not shown), such as speakers and printers.

[0060]   The computer system 500 may operate in a networked environment using logical connections to one or more remote computers, such as a remote computer 546. The remote computer 546 may be a computer system, a server, a router, a network PC, a peer device or other common network node, and typically includes many or all of the elements described above relative to the computer system 500. The network connections include a local area network (LAN) 548 and a wide area network (WAN) 550. Such networking environments are commonplace in offices, enterprise-wide computer networks, intranets, and the Internet.

[0061]   When used in a LAN networking environment, the computer system 500 is connected to the local network 548 through a network interface or adapter 552. When used in a WAN networking environment, the computer system 500 typically includes a modem 554 or other means for establishing communications over the wide area network 550, such as the Internet. The modem 554, which may be internal or external, is connected to the system bus 506 via the serial port interface 540. In a networked environment, program modules depicted relative to the computer system 500,

or portions thereof, may be stored in the remote memory storage device. It will be appreciated that the network connections shown are exemplary, and other means of establishing a communication link between the computers may be used.

[0062] In an embodiment of the present invention, the computer 500 represents a web server, wherein the processor 502 executes a page factory module on an ASP+ resource stored on at least one of storage media 516, 512, 514, 518, 519, or memory 504. HTTP responses and requests are communicated over the LAN 548, which is coupled to a client computer 546.

[0063] FIG. 6 illustrates a process flow diagram representing server-side processing of a page object and other control objects in an embodiment of the present invention. In operation 600, a page object constructor is called by the page factory module 308 (see FIG. 3). As a result, a page object (see e.g., the page object 314 in FIG. 3) is created to logically correspond to the client-side user interface element on the client. In operation 602, the page factory module calls the ProcessRequest member function of the page object, which initiates the staged operations for processing the HTTP request received from a client. In a first stage of one embodiment of the present invention, a server-side Create operation (not shown) creates the descendant server-side control objects contained in the control object hierarchy of the page object, that is, constructors for child control objects are recursively called to create the control objects during the lifetime of the processing of the HTTP request processing.

[0064] In an alternate embodiment, however, creation of child control objects is deferred until the control object is required for a given processing step (e.g., handling a postback event, handling postback data, loading or saving a viewstate, resolving data binding, or rendering HTML code for the corresponding user interface element). The latter embodiment, which is said to implement "deferred control object creation", is an optimization that can alleviate unnecessary CPU and memory utilization. For example, a user input event received from the client may result in the creation of a completely different web page. In this case, it is unnecessary to instantiate an entire control object hierarchy of the previous page only to process an event that immediately results in the termination of the control object hierarchy and the instantiation of a new and different control object hierarchy for a new page.

[0065] In response to the server call to the page object's ProcessRequest method, operation 604 through 620 may be executed by the page object and by individual descendant control objects, depending in part on the data of a given HTTP request. In an embodiment of the present invention, the operation 604-620 are performed for each individual object in the order illustrated in FIG. 6; however, a given operation for one object may occur out of order or not at all with respect to a given operation of another object, depending on the HTTP request. For example, a first object may perform its Init operation 604 and its Load operation 606, and begin postback data processing operation 608, before a descendant control object performs its own Init operation 604 and Load operation 606 by virtue of deferred control object creation. The order of operation processing by the page object and descendent control objects depends on various factors, including without limitation the nature of the data in the HTTP request, the configuration of the control object hierarchy, the current state of the control objects, and whether deferred control object creation is implemented.

[0066] The Init operation 604 initializes a control object after it is created by executing any server-side code associated with initialization in the dynamic content resource. In this manner, each server-side control object may be customized with specific server-side functionality that is declared in the dynamic content resource. In an embodiment of the present invention, dynamic content code intended to customize or extend the base page control classes as declared by the page developer in the ASP+ resource on the server. When the ASP+ resource is compiled, the declared code is included in the appropriate initialization code (e.g., the Init() methods of the page object and the descendent control objects). The Init operation 604 executes this code to customize or extend the page base class and the base classes for descendent control objects.

[0067] In an embodiment of the present invention, state management of the server-side control objects is supported in a Load operation 606 and a Save operation 616, which use a transportable state structure to accommodate the stateless model for client server systems by restoring server-side control objects to their previous states. In one embodiment, the state is communicated to and from the server in one or more hidden HTML fields of an HTTP request/response pair, although other transportable state structures are contemplated within the scope of the present invention.

[0068] In a given sequence of requests and responses relating to the current page between a client and a server, the states of one or more control objects are recorded into a transportable state structure by the Save operation 616 after the processing of a previous request. In an embodiment of the present invention, additional state information is also included in the transportable state structure, including hierarchical information or control object identifiers to allow the server to associate a given state with the appropriate control object. In a subsequent HTTP request, the state information is returned to the server in the transportable state structure. The server extracts the state information from the received transportable state structure and loads the state data into the appropriate control objects within the control object hierarchy to restore each control object to its state as it existed prior to a previous HTTP response. After the current request is processed, the states of one or more server-side control objects are again recorded into the transportable state structure by the Save operation 616, and the transportable state structure is returned to the client in the next HTTP response.

[0069] As a result of the Load operation 606, each server-side control object is placed in a state consistent with its state prior to a previous HTTP response. For example, if a text box control object includes a property value equaling "JDoe" prior to a previous HTTP response, the Load operation 606 restores the same control object to its previous state, in part by loading the text string "JDoe" into the property value. In addition, whether the state of a given object is stored and restored is configurable.

[0070] In summary of one embodiment of the present invention, the state of one or more server-side control objects is "saved" after processing. The saved state information is transmitted to the client in a response. The client returns the saved state information to the server in a subsequent response. The server loads the state information a freshly instantiated server-side control object hierarchy, such that the state of the hierarchy is restored to its previous state.

[0071] An alternative embodiment may maintain the state information on the server or at some other web location accessible by the server during the round trip from the server to the client, and then back to the server. After the client request is received by the server, this state information may be retrieved by the server and loaded into the appropriate server-side control object(s) in the control object hierarchy.

[0072] In operation 608, postback data received from the HTTP request is processed. Postback data may be included in the payload of the HTTP request in key-value pairs, in a hierarchical representation (e.g., XML), or in other data representations, such as RDF ("Resource Description Framework"). Operation 608 parses the payload to identify a unique identifier of a target server-side control object. The target server-side control object can be nested within an arbitrary hierarchy. If the identifier (e.g. "page1:list1:listrow2:text1") is found and the corresponding target server-side control object exists in the control object hierarchy, the received postback data is passed to the control object. For example, referring to FIG. 1, a unique identifier associated with textbox 106 and the text "JDoe" are communicated in the payload of the HTTP request 114 to the web server 116. Operation 608 parses the payload of the HTTP request 114 and obtains the unique identifier of the textbox 106 and its associated value (i.e., "JDoe"). Operation 608 then resolves the unique identifier of the textbox 106 to identify the corresponding target server-side control object and passes the "JDoe" value to the target object for processing.

[0073] As discussed with regard to the Load operation 606, the property values of server-side control objects may be restored to their previous states. In response to the receipt of postback data, the server-side control object determines whether the passed-in value causes a change from its corresponding property's previous value. If so, the change is logged in a change list to indicate a data change for the associated control object. After all postback data has been processed within the control object hierarchy, a call may be made to a control object method to raise one or more postback data changed events to one or more non-user-interface server components, such as a stock price look-up application running on the server. An example of a postback data changed event is an event indicating that postback data has causes a property of a server-side control object to change. In an exemplary embodiment, such an event can be sent to a system-provided event queue so that server-side code, which may be registered to process the event, can be invoked. The server-side code may then call a method of the non-user-interface server component. In this manner, the server-side non-user-interface server component can respond to events triggered by a change in data of a control object. Alternative methods of implementing events are also contemplated in the scope of the present invention, including using application-provided event queues, polling, and processing interrupts.

[0074] In operation 610, postback events are handled. In one embodiment, a postback event is communicated to the server in an event structure recorded in a query string. For example, at least two intrinsic HTML tag elements - "<input type=submit>" and "<input type=image>" - cause their containing form to be submitted to the server when the element is "clicked" by a mouse. Two exemplary button element are declared by the following HTML code:

```
<form action="Foo.aspx" method="GET">
    Name: <input type="text" name="scott">
    <input type=submit name="ButtonI"> I<input type=submit
name="Button2">
<\form>
```

[0075] If "Button1" is clicked, the query string "Foo.aspx?name=scott&ButtonI" is posted to the server. If "Button2" is clicked, the query string "Foo.aspx?name=scott&Button2" is posted to the server.

[0076] In an alternative embodiment, postback events are recorded known hidden fields in the payload of the HTTP request. The embodiment is particularly useful when an HTML element is not capable of generating form submits back to the server. Instead, a client-side script is invoked to populate an event structure in the payload of the HTTP request.

[0077] Operation 610 parses a specified event target identifying the server-side control object to which the event is directed. In one embodiment, the specified event is recorded in the query string. In another embodiment, the specified event is recorded in a hidden field in the HTTP request (e.g., labeled "_EVENTTARGET" in an embodiment of the present invention). Alternative embodiments may employ other event structures (e.g., visible fields, remote resources, etc.) to be parsed in operation 610.

[0078] Furthermore, in an embodiment of the present invention, operation 610 parses event arguments, if any, and provides the event arguments (e.g., labeled "_EVENTARGUMENT" in an embodiment of the present invention) to the

specified server-side control object. The server-side control object raises its events for processing by another server-side control object or by other server-side code, which can call a method of a non-user-interface server component (e.g., a server-side stock price look-up application) associated with the dynamic content resource.

[0079]   Operation 612 resolves data binding relationships between the server-side control objects and one or more databases accessible by the server, thereby updating in control object properties with database values and/or updating database fields with values of control object properties. In an embodiment of the present invention, properties of server-side control objects may be associated (or data bound) to properties of a parent data binding container, such as a table in a server-side application database. During the data binding operation 612, the page framework may update a data bound control object property with the value of the corresponding parent data binding container property. In this manner, user interface elements on the web page of the next response accurately reflect updated property values, because the control object properties to which the user interface elements correspond have been automatically updated during the data binding operation 612. Likewise, control object properties can also be updated to the parent data binding container fields, thereby updating a server-side application database with postback input from a server-side control object.

[0080]   Operation 614 performs miscellaneous update operations that may be executed before the control object state is saved and the output is rendered. Operation 616 requests state information (i.e., viewstate) from one or more control objects in the control object hierarchy and stores the state information for insertion into a transportable state structure that is communicated to the client in the HTTP response payload. For example, a "grid" control object may save a current index page of a list of values so that the "grid" control object may be restored to this state after a subsequent HTTP request (i.e., in operation 606). As described above, the viewstate information represents the state of the control object hierarchy prior to any subsequent actions by the client. When the viewstate information is returned, it will be used to place the control object hierarchy in that previous state prior to processing any client postback input or databinding.

[0081]   The render operation 618 generates the appropriate authoring language output (e.g., HTML data) for communication to the client in an HTTP response. Rendering is accomplished through a top-down hierarchical tree walk of all server-side control objects and embedded rendering code. Operation 620 performs any final cleanup work (e.g., closing files or database connections) before the control object hierarchy is terminated. Processing then returns to operation 602 and proceeds to operation 622 where the page object is terminated by calling its destructor.

[0082]   FIG. 7 illustrates a flow diagram of client-side operations relating to processing postback input (e.g., postback data and postback events) in an embodiment of the present invention. When a user interacts with a web page at a client, the user is often presented with client-side user interface elements, such as text boxes, buttons, scroll boxes, etc., in the displayed web page. In an embodiment of the present invention, some client-side user interface elements accept client-side data input or raise client-side events. A client-side text box in a web page, for example, typically receives text data input by a user (see the text data "JDoe" entered into the text box 106 of FIG. 1) for transmission to a web server in an HTTP request. Such transmitted data is referred to herein as "postback data".

[0083]   The client-side text box can also be configured to raise a client-side event. For example, a client-side "data changed event" can be raised if the text data entered therein causes the text box data property or value to change from its previous value on the client. A client-side event handler, which may be written in a scripting language, may be called to process the client-side event. Client-side events can, but need not, result in a postback event. In an embodiment of the present invention, the client-side event handler can invoke a postback event by populating event structures (e.g., _EVENTTARGET and _EVENTARGUMENT or the query string) associated with the appropriate user interface element and calling a submit() method. The submit() method causes postback data and postback events (i.e., in the form of the event parameters) to be inserted into the payload of an HTTP request, which is transmitted to the web server. While the exemplary user interface element in this discussion was a text box, other user interface elements also generate postback data or postback events including, without limitation, list boxes, buttons, check boxes, and scroll boxes.

[0084]   Postback data and events may be communicated from the client to the web server in an HTTP request. The format of an exemplary HTTP request consists of:

•   an initial line;
•   zero or more header lines;
•   a blank line (e.g., a carriage return/line feed character); and
•   an optional message body or payload.

[0085]   As defined in the HTTP standard, the initial request line in an HTTP request includes three parts, separated by spaces: (1) an HTTP method name; (2) the local path of the requested resource (e.g., an HTML file, a ASP+ resource, etc.), and (3) the version of HTTP being used. The local path is also called a request URI ("Uniform Resource Identifier"), which is a generic term referring to various types of names and addresses that refer to objects on the worldwide web.

A URL is a kind of URI. Possible HTTP methods include without limitation GET, HEAD, and POST. The GET method indicates that the specified resource is being requested and presents all postback data to the web server in the environment variable QUERY_STRING in the HTTP request payload. The HEAD method is similar to a GET method, except it asks the server to return the response headers only, and not the actual resource (i.e., no message body). The POST method indicates that the specified resource is being requested and presents all postback data to the web server in the HTTP request payload.

[0086]   In an embodiment of the present invention, postback data and postback events are communicated to the web server using key-value pairs that follow a unique naming convention (e.g., the postback input values are associated with unique hierarchical keys). The hierarchical key (e.g., "/Page1/List1/ListRow2") identifies an associated server-side control object corresponding to the client-side user interface that sourced the postback input. A postback data value or event argument is then associated with the unique key. In this manner, the web server can distribute the postback data and events to the appropriate server-side control object in the control object hierarchy.

[0087]   In the illustrated flow of FIG. 7, waiting operation 700 waits for a client-side event, which may include, without limitation, a "data changed event" (caused by a change in a data value of a client-side user interface element), a button click, a scroll operation, etc. In detecting operation 702, the client-side event is detected by a handler. In handling operation 704, a client-side handler (e.g., a client-side script module) is invoked to handle the client-side event. In processing operation 706, the event handler performs appropriate processing, such as storing new input data into form data values. For client-side events relating to postback data changes, the form data values associated with the given user interface element are updated.

[0088]   For client-side events relating to postback events, the event structures, such as _EVENTTARGET and _EVENTARGUMENT, associated with a given server-side control object are updated. It should be understood that hidden fields are employed in an exemplary embodiment of the present invention, although other methods of communicating event information may be employed, including as the query string, visible fields and other attributes of the client's request.

[0089]   In an alternative embodiment, the postback input may be transmitted in the form data of the HTTP request, without being recorded into hidden fields in the HTTP request by a client-side script (e.g., operations 704 and 706 may be bypassed). Instead, a standard "submit" command cause an HTTP "post" method, inserting into the form data one or more key-value pairs corresponding to properties of client-side user interface elements.

[0090]   Even if the client-side event is related to an eventual postback event, the client-side event need not result in a postback event. Referring to decision operation 708, if the client-side event handler does not result in a submit() call, processing returns to waiting operation 700 to wait for another client-side event. Otherwise, processing proceeds to submitting operation 710 where postback data and events are combined into the payload of an HTTP request and transmitted to the web server. After the web server has completed processing the HTTP request, it sends an HTTP response with freshly rendered authoring language data (received in receiving operation 712) for display in a web page on the client in displaying operation 714.

[0091]   FIG. 8 illustrates a flow diagram of server-side operations for processing postback input in an embodiment of the present invention. In receiving operation 800, the web server receives the HTTP request transmitted from the client in submitting operation 710 of FIG. 7. A creating operation 802 instantiates the appropriate page object associated with the resource specified in the URL of the HTTP request.

[0092]   In an embodiment of the present invention, the creating operation 802 also results in the creation of all other server-side control objects in the control object hierarchy. In another embodiment, the page object is created in the creating operation 802, while descendant control objects are created only when needed to perform a subsequent operation. The latter is referred to as "deferred control object creation".

[0093]   In initiating operation 804, one or more server-side control objects are initiated using a call to the "Init()" method. In operation 806, the state of one or more control objects, as received in an HTTP request containing a transportable state structure, is loaded into one or more server-side control objects to restore the control objects to their previous state.

[0094]   Examining operation 810 examines the incoming postback data in the HTTP request to find and extract identifiers ("IDs") corresponding to server-side control objects in the control object hierarchy. An identifier is generally found as a key in a key-value pair in the form data section of the HTTP request. Locating operation 812, having found a server-side control object ID, attempts to locate the corresponding server-side control object within the hierarchy. Locating operation 812 is described in more detail with regard to FIG. 9. Having found the corresponding server-side control object, loading operation 814 passes the associated postback data into the located server-side object. If loading operation 814 results in a change to the data previously stored in the server-side control object (i.e., previous data), data change operation 816 indicates a data change associated with the server-side control object. In an embodiment of the present invention, the previous data stored by the server-side control object is compared with the postback data to be stored, and, if the postback data differs from the previous data, a data change is indicated and the previous data is replaced with the postback data in the server-side control object. Alternative methods of indicating a data change

and replacing the previous data with the postback data are also contemplated in the scope of the present invention.

[0095] In an embodiment of the present invention, a data change indication is performed by recording in a list an identifier of the control object for which the data change was detected. The data change indications may stored in an array or linked list, although other storage configurations are contemplated within the scope of the present invention. Decision operation 818 directs processing to the examining operation 810 if more postback data is available. In this manner, the actual raising of the data change events is deferred until postback data for other control objects has been input and the data change indication for such control objects is recorded into the list. After no other postback data remains to be processed, decision operation 818 directs processing to raise event operation 820 to ensure that the control objects in the hierarchy are in a stable state before individual data change events are raised.

[0096] The calling operation 820 calls a method to raise a server-side data change event for each control object associated with an indication of a data change. In event-raising operation 822, the server-side events are raised to server-side code that calls a method of an non-user-interface server component associated with the dynamic content resource. As such, data change events on a web page can be detected on the server to trigger various server-side data change events. These server-side events can then be detected by server-side code that can cause various processing tasks, such as updating orders in a customer database, etc. For example, a data change event may be raised to server-side code that calls a method of an e-commerce application to indicate a change in a consumer's selection in an online shopping web page.

[0097] After postback data has been processed in operations 810 through 822, postback events are processed in operations 824 through 832. Examining operation 824 examines the received HTTP request to locate and extract IDs for server-side control objects within the control object hierarchy. Locating operation 826 finds the control object that is the event target in the control object hierarchy. More details of locating operation 826 are shown with regard to FIG. 9. Calling operation 828 calls a method in the located control object to raise a server-side event. Also, the calling operation 828 may pass the event argument to the called method.

[0098] Event-raising operation 830 raises a server side event to server-side code in response to the call of calling operation 828. The server-side code can then call a method of a non-user-interface server component in response to the raised event. Furthermore, the server side code can interpret or filter raw event data received from the client and pass only the relevant data to the non-user-interface server component. For example, if a user clicks on an client-side image element, the postback event contains information including the X and Y coordinates of a user's click. The coordinates are stored as properties in the corresponding server-side control object of the image. In response to the raising of the click event on the server, the server-side code can extract the X and Y properties from the server-side control object and pass them to the non-user-interface component. As such, the developer of the non-user-interface component need not program the code required to extract the needed coordinates from the server-side control object.

[0099] Decision operation 832 directs processing to examining operation 824 if more postback events are included in the HTTP request. Otherwise, processing proceeds to calling operation 834 where a PreRender() method is called to invoke any other processing that is to occur before the generation of authoring language data. Calling operation 836 calls a Save() method to save the current state of control objects in the control object hierarchy. Calling operation 838 calls a Render() to generate authoring language data used to display a web page on the client. Calling operation 840 calls a Dispose() method. Responding operation 842 generates an HTTP response for communication to the client.

[0100] FIG. 9 illustrates a flow diagram of server-side operations for locating a server-side control object in a control object hierarchy, in response to a receipt of a control object hierarchical identifier ("ID"), in an embodiment of the present invention. Receiving operation 900 receives an ID, which was received in an HTTP request from a client. In an embodiment of the present invention, the ID has a hierarchical naming structure (e.g., as a hierarchical path) to uniquely identify individual control objects within the control object hierarchy. Each level of the identifier is referred to as a "node level identifier". The lowest level of an ID identifies a control object that is a leaf node of the ID. For example, given an ID of "/Page1/List2/ListRow1", the top node level identifier of the ID is "Page1", "List2" and "ListRow1" are node level identifiers, and "ListRow1" represents a leaf node of the ID.

[0101] Reading operation 902 parses the top node level identifier of the ID. Decision operation 904 determines whether a control object exists with the parsed node level identifier. If so, decision operation 906 detects whether the current control object is a leaf node of the ID. If the current control object is not a leaf node of the ID, reading operation 908 parses the next level of the ID and processing proceeds to decision operation 904. Otherwise, if the control object is a leaf node of the ID, returning operation 910 returns the control object for use in processing given postback data or a given postback event.

[0102] If decision operation 904 does not locate a control object that exists (i.e., is already created) with the current node level identifier, decision operation 912 determines whether the current object possesses a non-created child corresponding to the node level identifier. If not, an error is issued in error operation 914 because the ID did not correspond to any existing or prospective control object in the control object hierarchy. Otherwise, deferred creation operation 916 creates non-created children of the current control object, resulting in "deferred control object creation". Processing then proceeds to returning operation 906.

[0103] The embodiments of the invention described herein are implemented as logical steps in one or more computer systems. The logical operations of the present invention are implemented (1) as a sequence of processor-implemented steps executing in one or more computer systems and (2) as interconnected machine modules within one or more computer systems. The implementation is a matter of choice, dependent on the performance requirements of the computer system implementing the invention. Accordingly, the logical operations making up the embodiments of the invention described herein are referred to variously as operations, steps, objects, or modules.

[0104] The above specification, examples and data provide a complete description of the manufacture and use of the composition of the invention. Since many embodiments of the invention can be made without departing from the spirit and scope of the invention, the invention resides in the claims hereinafter appended.

Claims

1. A method of performing server-side processing of postback input received from a client and associated with a client-side user interface element, the method comprising:

    examining the postback input to determine an identifier of a target server-side control object;
    identifying the target server-side control object based on the identifier of the target server-side control object;
    passing the postback input to the target server-side control object; and
    processing the postback input passed to the target server-side control object.

2. The method of claim 1 wherein the processing operation comprises changing a property of the target server-side control object; and further comprising:
    generating authoring language data from the target server-side control object based on the property to define the client-side user interface element for transmission to the client.

3. The method of claim 1 or 2 wherein the processing operation comprises raising a server-side event from the target server-side control object; and further comprising:
    generating authoring language data from the target server-side control object based on the server-side event to define the client-side user interface element for transmission to the client.

4. The method of any of claims 1 to 3, in particular claim 2 further comprising:

    creating a plurality of server-side control objects in a server-side control object hierarchy, prior to the operation of processing the postback input; and
    terminating the plurality of server side control objects, after the operation of generating authoring language data.

5. The method of any of claims 1 to 4, in particular claim 2 further comprising:

    searching for the target server-side control object in a server-side control hierarchy based on the identifier;
    creating the target server-side control object in the server-side control hierarchy, if the target server-side control object is not found by the searching operation; and
    terminating the server-side control hierarchy, after the operation of generating authoring language data.

6. The method of any of claims 1 to 5, in particular claim 1 wherein the identifier has a hierarchical identifier structure indicating a plurality of levels in a server-side control object hierarchy including a plurality of member server-side control objects, and the operation of identifying the target server-side control object comprises:

    extracting a node level identifier from the identifier;
    passing the node level identifier to a member server-side control object corresponding to the node level identifier;
    identifying the member server-side control object as the target server-side control object, if the node level identifier identifies a leaf node of the identifier;
    extracting a next node level identifier from the identifier of the target server-side control object, if the node level identifier does not identify a leaf node of the identifier, wherein the next node level identifier identifies a child server-side control object of the member server-side control object; and
    performing recursively the passing and identifying operations and the operation of extracting a next node level

identifier using the next node level identifier as the node level identifier and the child server-side control object as the member server-side control object, if the node level identifier does not identify a leaf node of the identifier.

7. The method of any of claims 1 to 6, in particular claim 1 wherein the operation of processing the postback input comprises:
storing a postback data value as a property the target server-side control object.

8. The method of any of claims 1 to 7, in particular claim 7 wherein the target server-side control object initially stores an old data value as a property, and the operation of storing a postback data value comprises:

associating the postback data value with the property;
indicating a data change associated with the target server-side control object, if the postback data value passed to the target server-side control object is different than the old data value of target server-side control object; and
replacing the old data value with the postback data value in the target server-side control object.

9. The method of any of claims 1 to 8, in particular claim 8 further comprising:
raising a server-side data change event after the operation of replacing of the old data value, if a data change is indicated.

10. The method of any of claims 1 to 9, in particular claim 7 wherein the target server-side control object is one of a plurality of member server-side control objects in a server-side control object hierarchy, and the operation of storing a postback data value comprises:

storing postback data values for all of the member server-side control objects in the server side control object hierarchy; and
raising at least one server-side data change event after the operation of storing a postback data value for all member server-side control objects, if at least one data change is indicated.

11. The method of any of claims 1 to 10, in particular claim 10 further comprising:

receiving the server-side data change event from the target server-side control object; and
invoking a function of a non-user-interface server component, based on the server-side data change event.

12. The method of any of claims 1 to 11, in particular claim 1 wherein the operation of processing the postback input comprises:
processing a postback event using the target server-side control object.

13. The method of any of claims 1 to 12, in particular claim 12 wherein the operation of processing a postback event comprises:

extracting from the postback input a postback event argument associated with the identifier;
passing the postback event argument associated with the identifier to the target server-side control object; and
processing the postback event argument using the target server-side control object.

14. The method of any of claims 1 to 13, in particular claim 13 wherein the operation of processing a postback input further comprises:
raising a server-side event from the target server-side control object, responsive to the operation of processing the postback event argument.

15. The method of any of claims 1 to 14, in particular claim 14 further comprising:

receiving the server-side event from the target server-side control object; and
invoking a function of a non-user-interface server component, based on the server-side event.

16. A computer data signal embodied in a carrier wave by a computing system and encoding a computer program for executing a computer process performing server-side processing of postback input received from a client and associated with a client-side user interface element, the computer process comprising:

examining the postback input to determine an identifier of a target server-side control object;
identifying the target server-side control object based on the identifier of the target server-side control object;
passing the postback input to the target server-side control object; and
processing the postback input passed to the target server-side control object.

17. A computer program storage medium readable by a computer system and encoding a computer program for executing a computer process performing server-side processing of postback input received from a client and associated with a client-side user interface element, the computer process comprising:

examining the postback input to determine an identifier of a target server-side control object;
identifying the target server-side control object based on the identifier of the target server-side control object;
passing the postback input to the target server-side control object; and
processing the postback input passed to the target server-side control object.

18. A computer program product for processing postback input received from a client and associated with a client-side user interface element, the computer process comprising:

creating a plurality of server-side control objects in a server-side control object hierarchy, the server-side control object hierarchy including a target server-side control object associated with the client-side user interface element;
passing the postback input to the target server-side control object;
processing the postback input received by target server-side control object; and
generating authoring language data from the plurality of server-side control objects to define a web page for display on a client.

19. The computer program product of claim 18 wherein the computer process further comprises:

receiving an identifier associated with the target server-side control object; and
identifying the target server-side control object within the server-side control object hierarchy, based on the identifier.

20. The computer program product of claim 18 or 19 wherein the computer process further comprises:
terminating the plurality of server side control objects, after the operation of generating authoring language data.
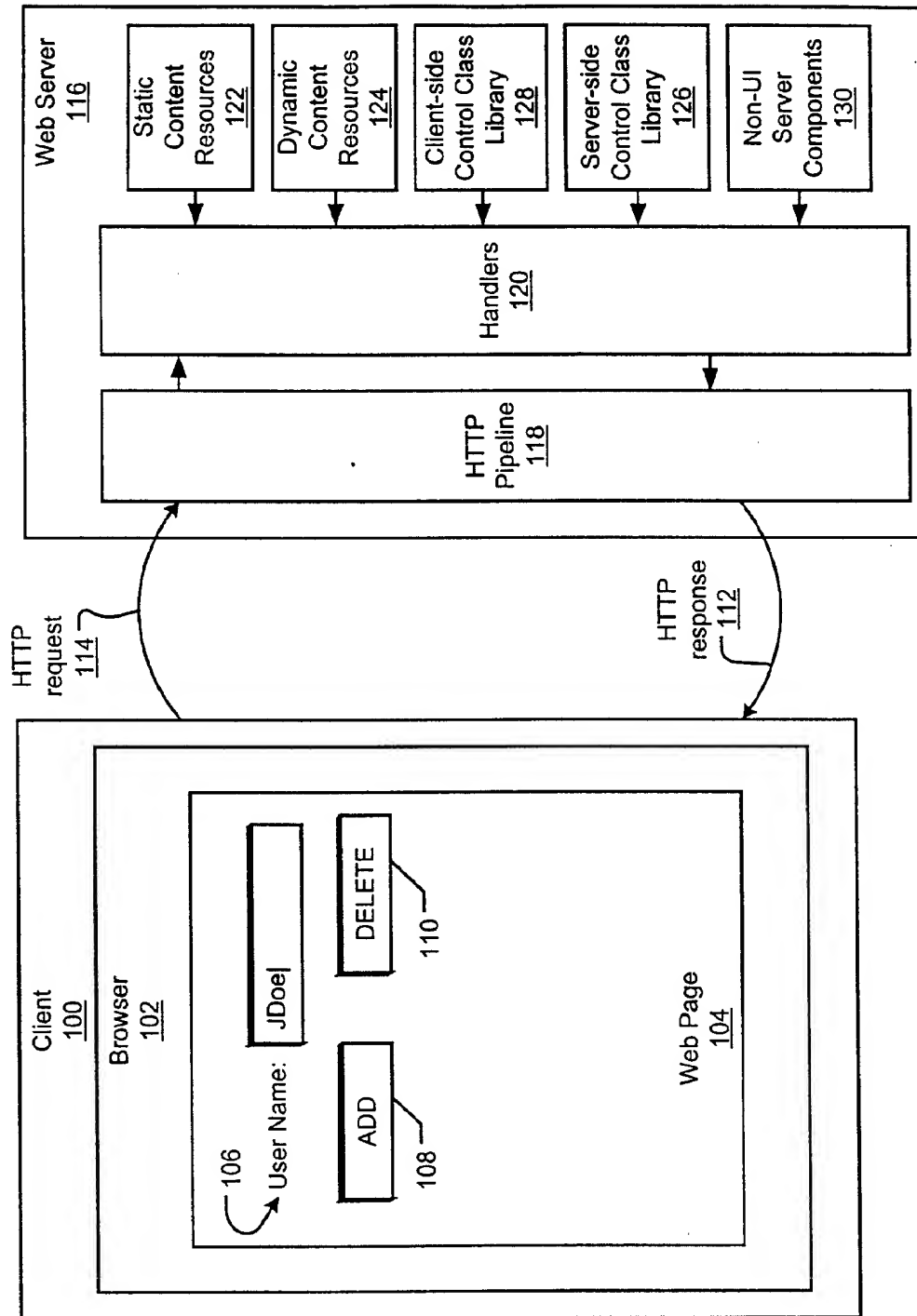
FIG. 1

Client

Server
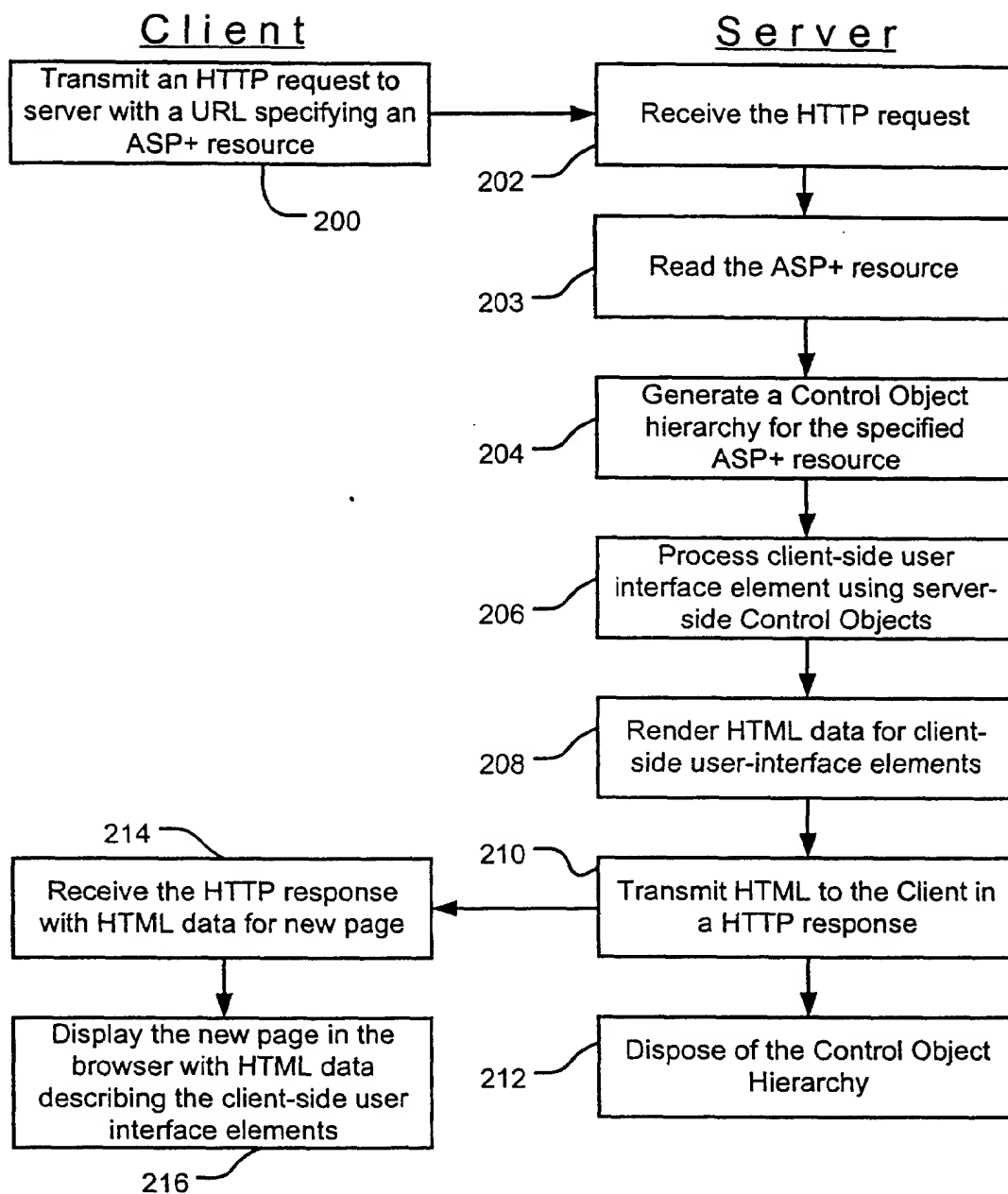
Transmit an HTTP request to
server with a URL specifying an
ASP+ resource

200

Receive the HTTP request

202

Read the ASP+ resource

203

Generate a Control Object
hierarchy for the specified
ASP+ resource

204

Process client-side user
interface element using server-
side Control Objects

206

Render HTML data for client-
side user-interface elements

208

214

Receive the HTTP response
with HTML data for new page

210

Transmit HTML to the Client in
a HTTP response

Display the new page in the
browser with HTML data
describing the client-side user
interface elements

216

Dispose of the Control Object
Hierarchy

212

FIG. 2
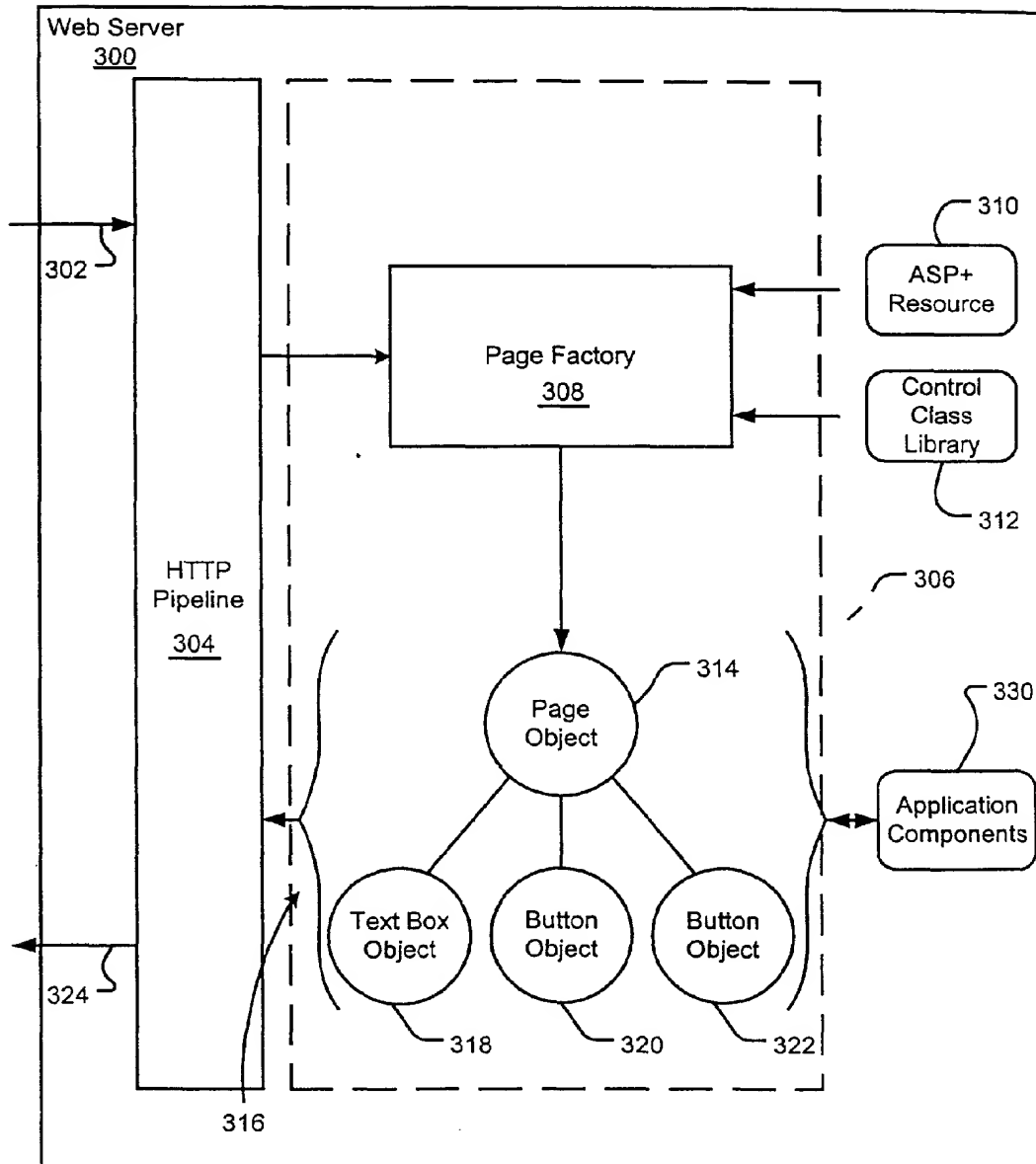
FIG. 3

```
1    <%@ Page Language="VB" Description="Simple Sample Page" Errorpage="ErrorPage.aspx" %>

2    <html>
3    <script runat=server>
4       Sub AddButton_Click(ByVal Source as Object, By Val E as Event Args)
5          Message.Text = "Add" & UserName.Text
6       End Sub

7       Sub DeleteButton_Click(ByVal Source as Object, By Val E as Event Args)
8          Message.Text = "Delete" & UserName.Text
9       End Sub
10   </script>

11   <body>
12   <form runat="server">
13       User Name:          <input type="Text" id="UserName" runat=server>
14       <br>
15       <button id="AddButton" value="ADD" OnServerClick="AddButton_Click" runat=server>
16       <button id="DeleteButton" value="DELETE" OnServerClick="DeleteButton_Click" runat=server
17       <br><br>

18       <span id="Message" runat=server> </span>
19   </form>
20   </body>
21   </html>
```
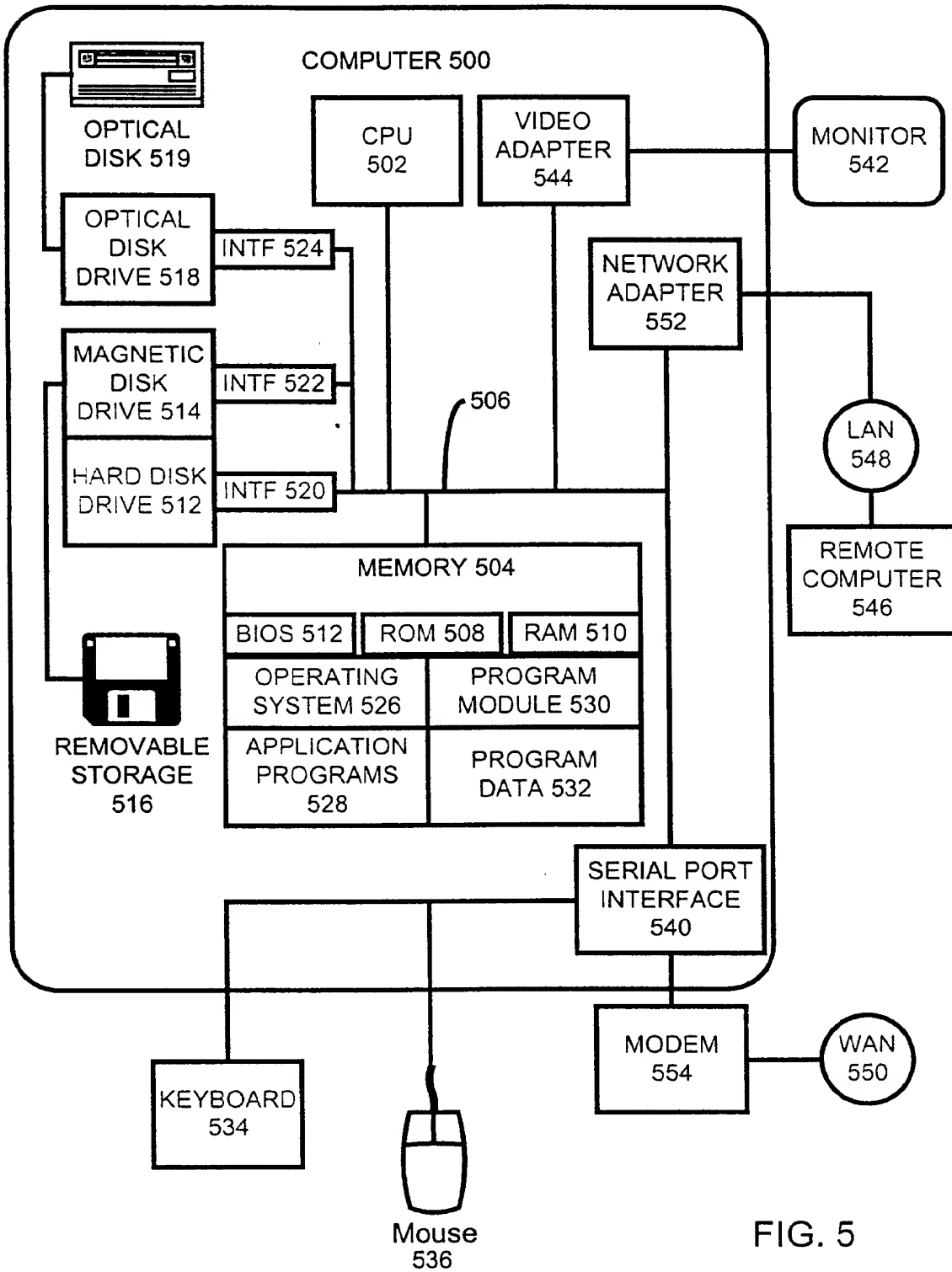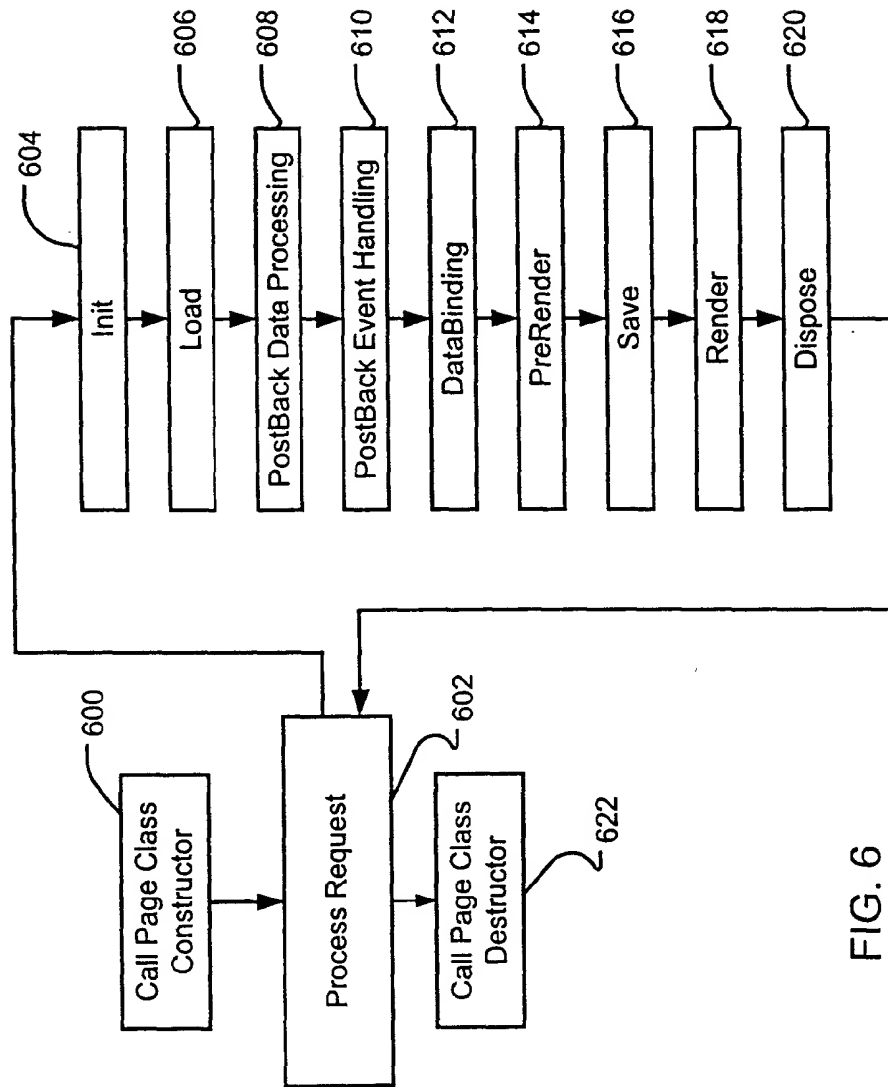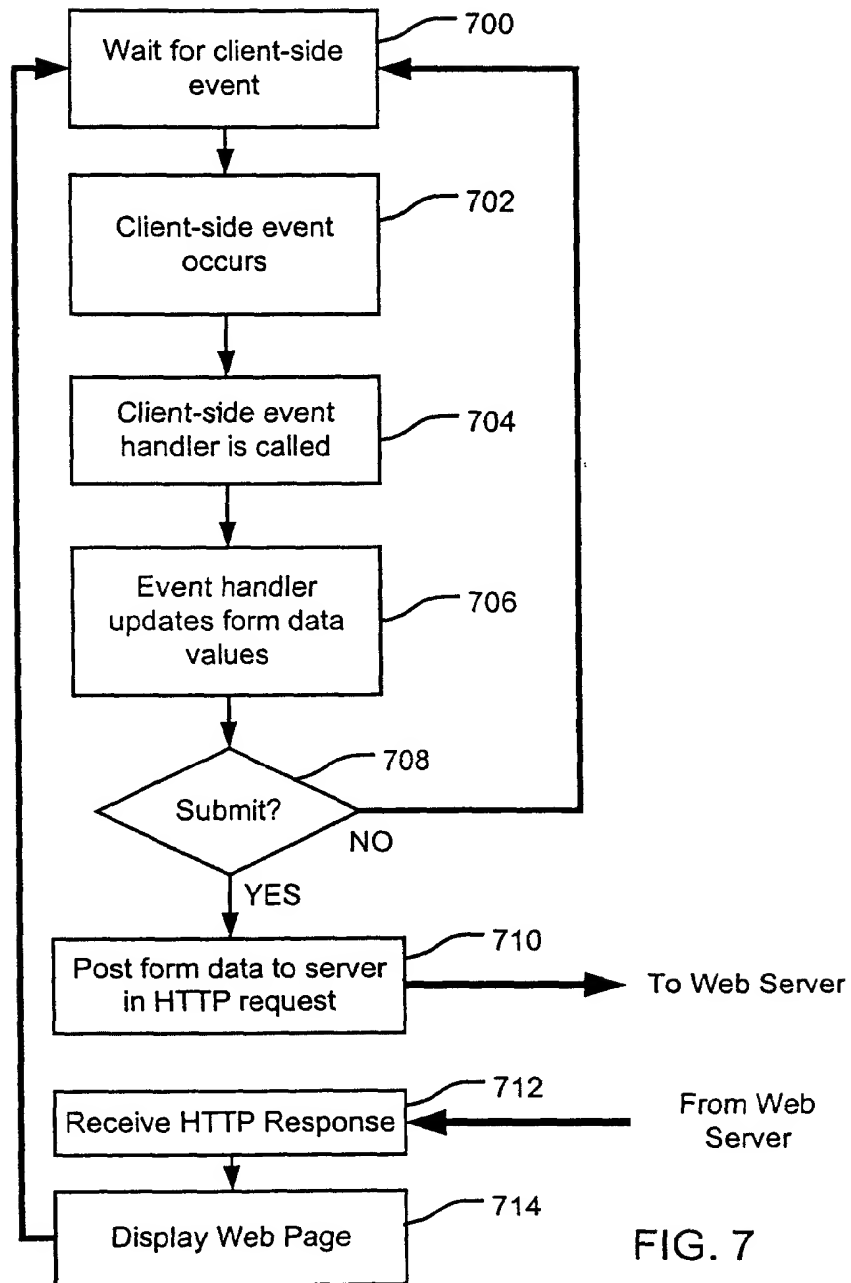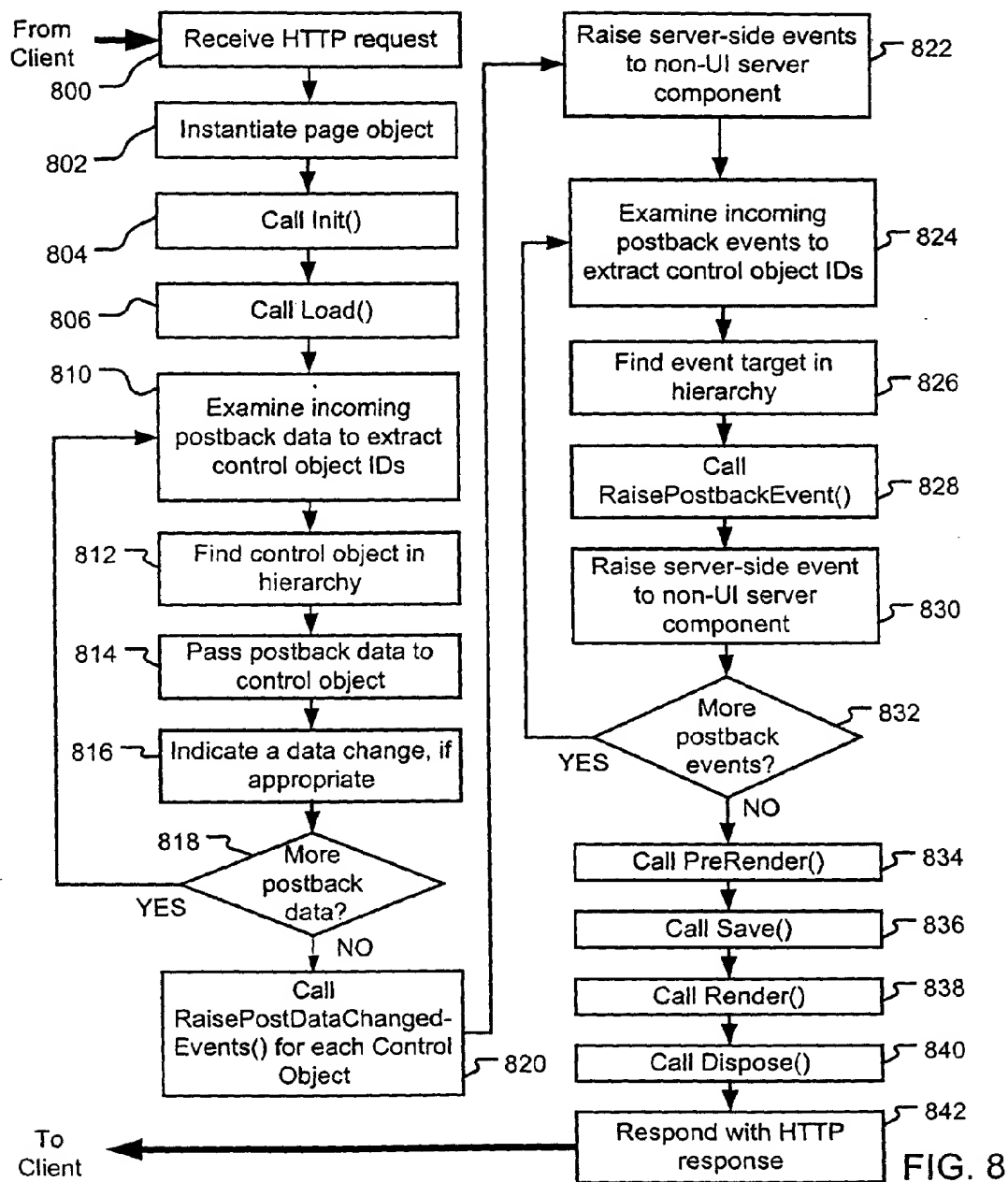
400

FIG. 4

COMPUTER 500

OPTICAL DISK 519

OPTICAL DISK DRIVE 518 — INTF 524

MAGNETIC DISK DRIVE 514 — INTF 522

HARD DISK DRIVE 512 — INTF 520

506

REMOVABLE STORAGE 516

CPU 502

VIDEO ADAPTER 544

MONITOR 542

NETWORK ADAPTER 552

LAN 548

REMOTE COMPUTER 546

MEMORY 504

| BIOS 512 | ROM 508 | RAM 510 |

OPERATING SYSTEM 526

PROGRAM MODULE 530

APPLICATION PROGRAMS 528

PROGRAM DATA 532

SERIAL PORT INTERFACE 540

KEYBOARD 534

Mouse 536

MODEM 554

WAN 550

FIG. 5

FIG. 6

FIG. 7

FIG. 8

FIG. 9